A Mean Field Model of Work Stealing in Large-Scale Systems

Nicolas Gast
Grenoble University and LIG
51, avenue Jean Kuntzmann
Montbonnont, France
nicolas.gast@imag.fr

Bruno Gaujal
INRIA and LIG
51, avenue Jean Kuntzmann
Montbonnont, France
bruno.gaujal@inria.fr

ABSTRACT

In this paper, we consider a generic model of computational grids, seen as several clusters of homogeneous processors. In such systems, a key issue when designing efficient job allocation policies is to balance the workload over the different resources.

We present a Markovian model for performance evaluation of such a policy, namely *work stealing* (idle processors steal work from others) in large-scale heterogeneous systems. Using mean field theory, we show that when the size of the system grows, it converges to a system of deterministic ordinary differential equations that allows one to compute the expectation of performance functions (such as average response times) as well as the distributions of these functions.

We first study the case where all resources are homogeneous, showing in particular that work stealing is very efficient, even when the latency of steals is large. We also consider the case where distance plays a role: the system is made of several clusters, and stealing within one cluster is faster than stealing between clusters. We compare different work stealing policies, based on stealing probabilities and we show that the main factor for deciding where to steal from is the load rather than the stealing latency.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; G.3 [**Probability and Statistics**]: Queuing theory

General Terms

Performance, Theory

Keywords

Mean Field, Grid Computing, Load Balancing.

1. INTRODUCTION AND RELATED WORK

A key issue when exploiting large-scale computer systems is to efficiently distribute the workload among the different resources. There are two main approaches to do so: *Push* strategies, in which a processor that is overloaded will send work to the others and *pull* strategies, in which an underloaded resource will ask for work from other resources.

Push strategies are mainly used in centralized systems where brokers allocate the jobs to the different processors. Although this technique is efficient for small systems, the resource allocation problem becomes difficult to solve when the system grows. Pull strategies are more appropriate for large-scale systems since they are distributed and oblivious of the speeds of the processors.

Work stealing is a strategy belonging to the *pull* category. The main principle is that when a resource becomes idle, it chooses another resource and *steals* part of its work. The choice of the resource to steal from depends on the implementation of work stealing. This technique is very efficient in practice and is implemented in various libraries, such as Cilk, Satin or Kaapi [9, 12, 22, 10]. This scheduling policy is very easy to implement and does not require any information on the system to work efficiently. Also, it is asymptotically optimal in terms of worst-case complexity [5]. Finally, it can be made *processor oblivious* since it automatically adapts to the number and to the size of jobs as well as to the speed of the processors [7].

A practical consideration for work stealing implementations is the stability of the algorithm. Such issues have been tackled in [6], where a system of homogeneous processors and a total arrival rate smaller than the total service rate is proved to be a positive recurrent Markov chain in a discrete time setting, the general case being still open.

Several models have been proposed to study the performance of work stealing. In [18], a continuous time Markov model of work stealing over a small number of processors is analyzed using a decoupling assumption. In [2], a numerical method based on a M/G/1 model of a work stealing system is presented. These two approaches do not scale with the number of processors and become untractable with more than 10 processors.

Here, we consider the case where the number of processors N is large. This is important in practice if work stealing is to be used in computational grids. We use mean field techniques to derive a system of ordinary differential equations that is the limit of a Markovian system when N goes to infinity. Such techniques have been used for the first time in [14] where the author derives differential equations for a system of homogeneous processors who steal a single job when idle. Here, we consider the case where, at every steal, half of the jobs in the queue of the victim are stolen. This strategy is closer to what is actually implemented in available work stealing libraries and is also much more efficient, as shown in the experimental section. It also makes the resulting model (both the Markov chain and the differential limit) more complicated because it contains non-local dependencies. Another important difference with [14] is the fact that we consider the case where the geometry of the system is taken into account as well as the heterogeneity of the processors. Communication times between processors in the same cluster are homogeneous, however stealing from a remote cluster takes more time and depends on the distance between the clusters. Finally, we also provide a new numerical technique that allows one to sample the distribution of the response times of one job, rather than just compute the average response time. This is more useful in practice because it provides guaranties to the users.

The paper is structured as follows. In Section 2, we describe a working stealing model in computational grids and the corresponding Markov process that falls in the category of density dependent population processes.

Section 3 discusses the convergence properties of the work stealing process when the number of processors goes to infinity. We provide the limit under the form of a set of deterministic ordinary differential equations (ODEs).

Section 4 focuses on the case where the system is made of a single cluster of homogeneous processors. We show that the ODEs have a single equilibrium point and we provide bounds on the speed of convergence to the mean field limit in extreme cases. A fast numerical method to compute the equilibrium point is given as well as a fast simulation algorithm to sample the distribution of the response times of jobs.

Finally, Section 5 extends the analysis to the case where processors are partitioned into several clusters. We study several scenarios (clusters with equal or unbalanced loads, master-slave cases) and we provide insight on the best stealing strategies. For example, is it worth stealing from remote clusters to balance the load at the expense of large stealing costs? So far, such issues have only been investigated experimentally in [17, 21] where real tests (with a small number of processors) have been conducted. Our paper shows that there is no universal answer to this question. Nevertheless, the rule of thumb is that the load factor is more critical to performance than the cost of stealing.

2. WORK STEALING IN GRIDS

2.1 Computational grids

The motivation of this work comes from computational grids. Such grids abound and have often replaced single supercomputers to provide high performance computing at a lower cost. The main architectural feature of computational grids is the fact that they are made of several clusters. Each cluster is composed of a large number of homogeneous processors, with homogeneous communications inside the cluster. However clusters are all different and the speed of communication between clusters depends on the couple of clusters considered. A typical example is Grid 5000 [1], an experimental grid deployed in Brazil, France and Luxemburg. The French part is made of 9 clusters spread all over the country, the most recent one being Digitalis (digitalis.inria.fr), a cluster of 700 Intel Nehalem processors over an InfiniBand internal network at 40 Gb/s. The clusters are inter-connected by a dedicated network at 10Gb/s. The parameters used in the paper all come from measurements made on Grid 5000.

Users of computational grids submit jobs to a selected cluster, or to a central broker. Jobs are allocated to processors by a *batch scheduler* (oar.imag.fr for grid 5000) whose role is to minimize the response times (also called sojourn times) of jobs *i.e.*, the time spent by the jobs in the system. The goal of this paper is to analyse the performance of such systems when a work-stealing strategy is used by the scheduler.

2.2 Work stealing model

Let us consider a model of a computational grid made of N processors. Here, a processor represents a generic computing unit such as a single CPU core, an entire CPU (*i.e.*, all cores in the CPU), an entire GPU, or a multi-CPU/GPU server. The processors are grouped into C clusters. Each cluster is composed of a large number of homogeneous processors with homogeneous communications inside the cluster. However, clusters are heterogeneous in size and processing speeds, and communications between clusters depend on their distance as well as the network type.

We consider that each processor in each cluster c receives jobs from outside. The jobs arrive according to a Poisson process of rate λ_c. Each job has a size that is exponentially distributed with mean 1. Actually, more general distributions can be taken into account, as long as they can be represented by finite Markov processes (such as Cox distributions). We further assume that tasks are independent and cannot be divided to be executed by two different processors. The processors in cluster c have a speed μ_c: the time taken by one of these processors to serve a job of size S is S/μ_c.

If a processor has more than one job to serve, it stores them in a buffer of maximal capacity K. If a job arrives when the buffer is full, it is discarded. We denote by $j_n(t)$ the number of jobs present in the nth processor. By definition, $j_n(t) = 0$ means that the processor has no job to serve and that its buffer is empty. Otherwise $j_n(t)$ is the number of jobs in its buffer plus one (corresponding to the job it is currently serving). From a queuing theory point of view, this means that all processors can be seen as $M/M/1/K$ queues.

If the processor i has no job to serve, it tries to steal some jobs from a different processor, called the victim. For that, it selects another processor, say k, at random (according to probabilities, defined later) and asks for jobs. This operation may take some time (exponentially distributed). If after this time the processor i is still idle and if k has $j_k \geq 2$ jobs, the jobs of processor k are shared between processors i and k: $\lfloor j_k/2 \rfloor$ for i and the rest stay in k. If i is in cluster c_i and k in cluster c_k, we consider that the time to wait for the answer is exponentially distributed of mean $1/\gamma_{c_i c_k}$. The selection of the victim is not completely uniform: the processor first selects a cluster according to the probability law $p_{cc'}$ and then picks uniformly a processor in this cluster. For simplicity, we neglect the time to transfer a job and only take into account the time to get an answer. This means in particular that the time to steal jobs does not depend on the number of stolen jobs but only on the localization of the two processors (*i.e.*, the clusters they belong to).

2.3 A Density dependent population process

A sequence of Markov processes X^N on $\frac{1}{N}\mathbb{N}^d$ ($d \geq 1$) is called a density dependent population process if there exists a finite number of transitions, say $\mathcal{L} \subset \mathbb{N}^d$, such that for each $\ell \in \mathcal{L}$, the rate of transition from X^N to $X^N + \ell/N$ is $N\beta_\ell(X^N)$, where $\beta_\ell(.)$ does not depend on N.

This model is often used to represent a population when the number of individuals goes to infinity. Each individual lives in a finite state space $\{1 \dots d\}$ and the ith component of the vector X^N represents the proportion of individuals in state i.

This model has been well studied in the literature. In particular, the work of Kurtz [8, 13] shows that as N grows, the behavior of the system goes to a deterministic limit. Moreover, this limit satisfies a set of ordinary differential equations that can be derived from the transition rates. These results are adapted to our case in Section 3.1. Here, we will show that our system can be described by a density dependent population process.

The state of a processor is given as follows. Let $\mathcal{C}$ be set of clusters $\mathcal{C} \overset{\text{def}}{=} \{1, \dots, C\}$ and $\mathcal{K}$ be the set of buffer sizes: $\mathcal{K} \overset{\text{def}}{=} \{0, \dots, K\}$. If the processor p belongs to cluster c_p and has $j_p \in \mathcal{K}$ jobs in its queue, its state is (c_p, j_p). Note that the cluster c_p of the processor p is fixed by the geometry of the system. If the processor p has no job in its queue, this means that it is trying to steal some jobs from another processor, say q. If p is in cluster c_p and q in cluster c_q, then the state of p is $(c_p, 0, c_q)$. Finally, $X^N_{c_p j_p}$ denotes the proportion of processors in state (c_p, j_p) and $X^N_{c_p 0 c_q}$ the proportion of processors in state $(c_p, 0, c_q)$.

Proposition 1. *$(X^N_{cj}, X^N_{c0c'})_{c,c' \in \mathcal{C}, j \in \mathcal{K}}$ is a continuous time Markov process on $\mathbb{R}^{CK+C^2}$. Moreover, the sequence of Markov processes X^N is a density dependent population process.*

Proof. Let us fix N and assume that at time t, the system is in state $X^N(t) = (X_{cj}(t) \dots X_{c0c'}(t))_{c,c' \in C, j \in \mathcal{K}}$. There are four types of events that can happen: arrivals, departures, successful steals and unsuccessful steals. Let t' be the time of the next event and let us compute the rate at which each event occurs.

If an *arrival* occurs on a processor of cluster c that has $1 \leq j \leq K - 1$ jobs, the corresponding modification of the state will be that $(X_{cj}, X_{c,j+1})$ will become $(X_{cj}(t) - 1/N, X_{c,j+1}(t) + 1/N)$. The jobs arrive according to a Poisson process of rate λ_c. This means that this transition occurs at rate $NX_{cj}(t)\lambda_c$. Similarly, an arrival on a processor of cluster c that has 0 jobs and is stealing from c' occurs at rate $NX_{c0c'}(t)\lambda_c$.

The case of *departures* is similar for processors that have $j \geq 2$ jobs. If a processor has 0 jobs, no departure is possible. When there is a departure from a processor of cluster c that has 1 job, this processor chooses a cluster c' to steal from (with probability $p_{cc'}$) and then a processor among them, uniformly. Therefore, $(X_{c1}, X_{c0c'})$ becomes $(X_{c1} - 1/N, X_{c0c'} + 1/N)$ at rate $NX_{c1}\mu_c p_{cc'}$.

Once a processor of cluster c is empty and has chosen a victim cluster, namely c', it asks for work to steal and gets its response with rate $\gamma_{cc'}$. If $X_{c'}$ is the proportion of processors in cluster c', this is equivalent to saying that the processor gets a response from each processor of cluster c' with a rate $\gamma_{cc'}/(NX_{c'})$. If the victim in cluster c' has j jobs, we distinguish two cases. If $j \geq 2$, the steal is *successful* and the processor gets $\lfloor j/2 \rfloor$ jobs. If $j = 0$ or 1, the steal is *unsuccessful* and the processor has to choose a new processor to steal from. Thus we can write the two following transitions:

- The *successful steal* of the jobs of a processor in cluster c' with j jobs from a processor in cluster c occurs with rate $N\gamma_{cc'}X_{c0c'}X_{c'j}/X_{c'}$ and changes $(X_{c0c'}, X_{c\lfloor j/2 \rfloor}, X_{c'j}, X_{c'\lceil j/2 \rceil})$ in $(X_{c0c'} - 1/N, X_{c\lfloor j/2 \rfloor} + 1/N, X_{c'j} - 1/N, X_{c'\lceil j/2 \rceil} + 1/N)$.
- The *unsuccessful steal* of a processor in cluster c trying to steal from cluster c' occurs when it steals 0 jobs. After that it chooses to steal from cluster c''. This event occurs with rate $N\gamma_{cc'}p_{cc''}X_{c0c'}(X_{c'0} + X_{c'1})/X_{c'}$ and changes $(X_{c0c'}, X_{c0c''})$ in $(X_{c0c'} - 1/N, X_{c0c''} + 1/N)$.

□

2.4 Examples

To illustrate the power of expression of our model, we present some examples that will be studied.

- **Homogeneous cluster** – in this case, all processors are homogeneous and each processor receives jobs at the same rate. This model is studied in detail in Section 4. We show that the steady state can be computed by a simple algorithm and we compute the main performance indicators.
- **Two homogeneous clusters** – we consider in Section 5.1 the case of two homogeneous clusters: they have the same parameters λ, μ. However, the rate of steal is 10 times larger inside a cluster than between clusters: $\gamma_{ii} = 10\gamma_{ij}$ if $i \neq j$.
- **Two heterogeneous clusters** – there is again two clusters and stealing is faster inside the cluster: $\gamma_{ii} = 10\gamma_{ij}$. The clusters are homogeneous in speed but one is more loaded than the other: $\lambda_2/\mu_2 > \lambda_1/\mu_1$. Section 5.2 studies the optimal stealing probability p_{ij}.
- **Master-Worker** – in this case, we consider a network of homogeneous clusters but the arrivals only occur in a fraction of the processors (called the masters). This is modeled by using two clusters per real cluster (one for the masters, one for the slaves) with the same parameters and with $\gamma_{ij} = \gamma_{ii}$ inside these two clusters. In Section 5.3, we study the performance of the system as a function of the fraction of masters in the system.

3. MEAN FIELD APPROXIMATION

In this section, we show that when the system grows large, its behavior can be approximated by a deterministic limit described by the system of ODEs (1-9). The following are the theoretical results that we use.

3.1 Convergence of density dependent population processes

In Section 2.3, we shown that our model can be described by a *density dependent population process.* Here we recall some of the convergence results for these processes. We refer readers to [8, 13] for a more complete description.

Let us call F the function $F(x) = \sum_{\ell \in \mathcal{L}} \ell \beta_\ell(x)$ (if the sum is well defined) and let us consider the following ordinary differential equation $x(0) = x_0$ and $\dot{x}_{x_0}(t) = F(x_{x_0}(t))$. Theorem 2 below shows that the stochastic process $X^N(t)$ converges to the deterministic system $x(t)$.

Theorem 2 ([8], chapter 11). *Assume that for all compact $E \subset \mathbb{R}^d$, $\sum_\ell |\ell| \sup_{x \in K} \beta_\ell(x) < \infty$ and F is Lipschitz* *on E. If $\lim_{N\to\infty} X^N(0) = x_0$ in probability, then for all $t > 0$:*

$$\lim_{N\to\infty} \sup_{s\le t} |X^N(s) - x(s)| = 0 \quad \text{in probability},$$

uniformly in the initial condition.

Moreover, Kurtz has proved second order results for the previous convergence, showing that the gap between $X^N(t)$ and $x(t)$ is of order $\sqrt{N}$. Let $V^N(t) \overset{\text{def}}{=} \sqrt{N}(X^N(t) - x(t))$, and $V(t) \overset{\text{def}}{=} V(0) + U(t) + \int_0^t \partial F(X(s))V(s)ds$, where $U(t)$ is a time inhomogeneous Brownian motion and ∂F denotes the Jacobian matrix of F.

THEOREM 3 ([8], CHAPTER 11). *Assume that for all compact set E: $\sum_\ell |\ell^2| \sum_{x\in E} \beta_\ell(x) < \infty$, that β_ℓ is Lipschitz, that F is continuously differentiable and $V^N(0) \xrightarrow{weak} V(0)$. Then for all t, $\sqrt{n}(X^N - X) \xrightarrow{weak} V$.*

This result indicates that for all fixed t, the gap between $X^N(t)$ and $x(t)$ behaves like $\frac{1}{\sqrt{N}}G$, where G is a Gaussian random variable.

3.1.1 Steady state convergence

Other interesting results of convergence concern the behavior of the steady state of the system. Assume for example that the stochastic process with N processors has an invariant measure, say π^N. A natural question is whether π^N converges (or not) to a fixed point of F and whether second order results exist in that case. In general, convergence for the steady state only holds under several restrictions.

THEOREM 4. *The support of any limit point (for the weak convergence) of the stationary distributions π^N is included in the compact closure of the accumulation points of the solutions of the equation $\dot{x} = F(x)$, for all possible initial conditions.*

The proof is inspired by the proof of Corollary 3.2 of [4].

PROOF. Let h be a continuous bounded function and $\delta > 0$. For a measure π, we denote $\int_x h(x)\pi^N(dx)$ the expectation of $h(X)$ if the distribution of X is π^N. $\mathbb{E}[X^N(t)]$ denotes the expectation over the trajectories of $X^N(t)$. Moreover, the system of differential equations $\dot{y} = F(y)$ starting in $y(0) = x$ has a unique solution. Its value at time t is denoted $\Phi_t(x)$.

Let π^N be an invariant measure of X^N and π a limit point of π^N. Since π^N is an invariant measure of X^N:

$$\int_x \mathbb{E}h(X^N(t))\pi^N(dx) = \int_x h(X^N(t))\pi^N(dx)$$

Since E is compact, h is uniformly continuous and there exists $\epsilon > 0$ such that $\|x - y\| < \epsilon$ implies $\|h(x) - h(y)\| < \delta$ and

$$\begin{aligned}\mathbb{E}\left|h(X^N(t)) - h(\Phi_t(X^N(0)))\right| \\ \le \delta + \epsilon \|h\| \mathbb{P}(\left\|X^N(t) - \Phi_t(X^N(0))\right\| > \epsilon)\end{aligned}$$

where $\|h\| \overset{\text{def}}{=} \sup_x \|h(x)\|$.

Theorem 2 shows that $\lim_{N\to\infty} \mathbb{P}(\left\|X^N(t) - \Phi_t(X^N(0))\right\| > \epsilon) = 0$ which shows that:

$$\lim_{N\to\infty} \left| \int_x \mathbb{E}h(X^N(t))\pi^N(dx) - \int_x h(\Phi_t(X^N(0)))\pi^N(dx) \right| \le \delta.$$

Using the fact that π^N goes weakly to π and since h and δ are arbitrary, this shows that π is an invariant measure of Φ_t. Let us call H the support of π and B the set of accumulation point of Φ_t. H is a closed invariant set and π is an invariant measure for Φ. By the Poincaré recurrence theorem, $\pi(H) = 1$. □

An immediate corollary is the case where the solutions of the differential equation all converge to a single global attractor.

COROLLARY 5. *If F has a unique stationary point x^* to which all trajectories converge, then the stationary measures π^N concentrate around x^* as N goes to infinity: $lim_{N\to\infty}\pi^N \xrightarrow{weak} \delta_{x^*}$, where δ_{x^*} is the Dirac measure in x^*.*

PROOF. Since the state space is compact, the sequence π^N is tight. By Theorem 4, it converges to δ_{x^*}. □

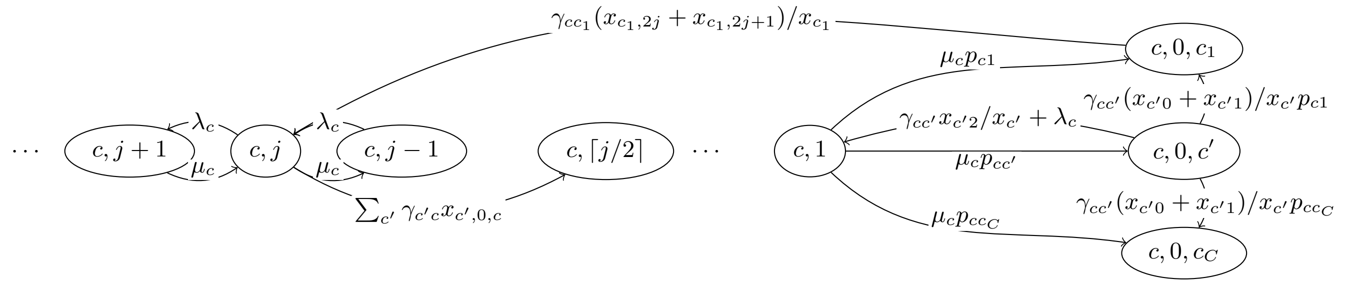
In the following we will show that the system of equations (1 - 9) has a unique fixed point. However, to apply Corollary 5, one needs to show that this point is a global attractor, which is a very difficult task for the set of equations (1 - 9) that do not admit a natural Lyapounov function.

In the case where a linear Lyapounov function exists, second order results for the steady state behavior exist. Norman [16] shows that under technical assumptions and if there exists a scalar product $\langle\cdot\rangle$ such that $\langle x - x^*, F(x)\rangle < 0$ and $\langle x, F'(x^*)x\rangle < 0$, then $\sqrt{N}(\pi^N - x^*)$ converges to a Gaussian variable. Unfortunately, we have not been able to construct such a scalar product for our case. However partial results on second order results are shown in Section 4.2.

3.2 Deterministic limit for work stealing

In Section 2.3 $(X^N_{c0c'}(t), X^N_{cj}(t))_{c,c'\in C, 1\le j\le K}$ is proved to be a density dependent population process. For each probability vector x on the state space $\{(c,j),(c,0,c')/c,c' \in C, j \ge 0\}$, let us recall the definition of the drift function $F(x) = \sum_\ell \ell\beta_\ell(x)$. Should Theorem 2 be used here, it would say that if $X^N(0)$ converges weakly to a deterministic measure $x(0)$, then $(X^N(s))_{0\le s\le t}$ converges weakly to the solution of the differential equation $\dot{x}(t) = F(x)$ over $[0,t]$. In our case, the system of ordinary differential equations $\dot{x}(t) = F(x)$ has the following form.

$$\begin{aligned}
\dot{x}_{c0c'} &= -(\lambda_c + \gamma_{cc'})x_{c0c'} + \mu_c x_{c1} p_{cc'} + &(1)\\
&\quad \sum_{c''} \gamma_{cc''} x_{c0c''} \frac{x_{c''0} + x_{c''1}}{x_{c''}} p_{cc'} &(2)\\
\dot{x}_{c1} &= -(\mu_c + \lambda_c)x_{c1} + \mu_c x_{c2} + \sum_{c'} \lambda_c x_{c0c'} &(3)\\
&\quad + \sum_{c'} \gamma_{c'c} x_{c'0c} x_{c2}/x_c &(4)\\
&\quad + \sum_{c'} \gamma_{cc'} x_{c0c'} (x_{c'2} + x_{c'3})/x_{c'} &(5)\\
\dot{x}_{cj} &= -(\mu_c + \lambda_c \mathbf{1}_{j<K})x_{c,j} + \mu_c x_{c,j+1} + \lambda_c x_{c,j-1} &(6)\\
&\quad + \sum_{c'} \gamma_{c'c} x_{c'0c} (x_{c,2j} + x_{c,2j-1})/x_c &(7)\\
&\quad + \sum_{c'} \gamma_{cc'} x_{c0c'} (x_{c',2j} + x_{c',2j+1})/x_{c'} &(8)\\
&\quad - \sum_{c'} \gamma_{c'c} x_{c'0c} x_{cj}/x_c, &(9)
\end{aligned}$$



$\gamma_{cc_1}(x_{c_1,2j}+x_{c_1,2j+1})/x_{c_1}$
$\cdots$
$c,j+1$
λ_c
μ_c
c,j
λ_c
μ_c
$c,j-1$
$c,\lceil j/2\rceil$
$\sum_{c'}\gamma_{c'c}x_{c',0,c}$
$\cdots$
$c,1$
$\mu_c p_{c1}$
$c,0,c_1$
$\gamma_{cc'}(x_{c'0}+x_{c'1})/x_{c'}p_{c1}$
$\gamma_{cc'}x_{c'2}/x_{c'}+\lambda_c$
$\mu_c p_{cc'}$
$c,0,c'$
$\gamma_{cc'}(x_{c'0}+x_{c'1})/x_{c'}p_{cc_C}$
$\mu_c p_{cc_C}$
$c,0,c_C$

Figure 1: Graph of the transition kernel of the state of one processor for the fast simulation algorithm. Due to the numerous transitions, only a fraction of the transitions are represented on this graph.

where $\mathbf{1}_{j<K}$ equals 1 for $j<K$ and 0 for $j\geq K$. Other boundary conditions are $x_{cj}=0$ for $j>K$.

These equations can be directly computed from the transitions described in Section 2.3. They can be interpreted as follows.

The first term in line 1 is the rate at which processors exit from state $(c,0,c')$: This happens if an arrival occurs (λ_c) or if a steal from c' occurs ($\gamma_{c0c'}$). The second term corresponds to the rate at which processors end up in state $(c,0,c')$. The line (2) of this equation represents the fraction of processors that were in cluster c trying to steal jobs from cluster c'' and that did not succeed and but decided to steal from the cluster c'. The following lines have a similar interpretation. For the last equation, line (7) represents the processors in cluster c that had $2j$ or $2j+1$ jobs and have been stolen by someone else and line (8), the processors from cluster c that are stealing from others. The last line (9) represents the processors in cluster c that had j jobs and have been stolen by someone else.

The two technical conditions for applying Theorem 2 are clearly satisfied: The function F in the differential equation is a rational function of degree 2 and is Lipschitz on all compacts. The transition set $\mathcal{L}$ is finite and the transition rate β_ℓ are bounded so that the second condition is also satisfied. Theorem 2 can be rewritten in this framework as:

Corollary 6. *Using, the foregoing notations, if $X^N(0)$ converges to $x(0)$ in probability, then $\sup_{0\leq t\leq T}|X^N-x(t)|\to 0$ in probability.*

3.3 Fast simulation algorithm

The previous theorem shows that the average number of processors in each state can be approximated by a system of differential equations. Here we show that as N grows large, the behavior on one particular processor can be approximated by a simpler process.

Let us consider a system with a finite number of processors $N<\infty$. Let $J^N(t)$ be the state of one particular processor at time t. It should be clear that the process $(J^N(t),X^N(t))$ is a continuous time Markov chain. For each population value x, we define the kernel $(K_{jj'}(x))_{j,j'\in\mathcal{C}\cup\mathcal{K}}$ as follows. If the population process $X^N(t)$ is x and $j,j'\in\mathcal{K}$, then $K_{jj'}(x)$ is the rate of transition of $J^N(t)$ from (c,j) to (c,j'). The definition for $j\in\mathcal{K}$ and $j'\in\mathcal{C}$ is similar, representing the rate of transition from (c,j) to $(c,0,j')$. The transition kernel $K(x)$ can be directly derived from the transitions written in Section 2.3 and are illustrated by Figure 1.

For finite N, the behavior of the processor $J^N(t)$ is not independent of the behavior of $X^N(t)$ as each transition of $J^N(t)$ will result in a change of $X^N(t)$. The process $J^N(t)$ is not Markovian and is very complicated. In the limit however, $J^N(t)$ goes to a non-homogeneous Markovian process.

Theorem 7. *Let us assume that $\lim_{N\to\infty}J^N(0)=y(0)$ and $X^N(0)\to x(0)$. Then $(J^N(t),X^N(t))$ converges weakly to a continuous time jump and drift process $(Y(t),x(t))$ where $x(t)$ satisfies the ODE (1-9) and $Y(t)$ is a non-homogeneous jump process of kernel $K(x(t))$.*

Proof. (sketch) This result is similar to Theorem 3.2.1 of [20] and can be proved using similar ideas. Conditionally to X^N, one can show that J^N is a non-homogeneous Markovian process with kernel $K(X^N(t))$. Since $X^N(t)$ converges in probability to $x(t)$, $\lim_{N\to\infty}K(X^N(t))=K(x(t))$. Finally, the convergence of J^N to J comes from Theorem 17.25 of [11]. $\square$

From a practical point of view, this theorem is important because it allows one to use the mean field approximation to compute distributions instead of average values. Moreover, the case of steady-state is of particular interest. Assume that the system of ODE has a unique stable point x^* to which all trajectories converge and that $X^N(0)$ is chosen according to the steady state distribution. In that case, $X^N(t)$ is distributed according to the steady state distribution and $\lim_{N\to\infty}X^N(t)=x^*$ (Corollary 5). Theorem 7 shows that the behavior of one processor chosen at random converges to a continuous time Markov chain of transition kernel $K(x^*)$. In Section 4.5, we will see how to use this result to compute the distribution of sojourn times.

4. ONE CLUSTER MODEL

In this part, we focus on the case where all processors belong to one homogeneous cluster. The system is described by 3 parameters: the arrival rate λ, the service rate μ and the rate of stealing γ.

Algorithms 1 and 2 provide very efficient ways to perform a steady state analysis. The total time to generate all curves of this section is less than ten minutes on a desktop computer.

4.1 Steady state limit

In this section, we show that the differential equations (1-9) without the boundary conditions ($j\leq K$) have a unique equilibrium point. We will also show that this relaxation is justified when $\lambda<\mu$, because the number of queues with more than j jobs decreases as α^j with $\alpha<1$. In the rest of this section, we assume that $\lambda<\mu$ to ensure stability.

An equilibrium point must satisfy the equation $\dot{x_j}(t) = 0$ for all $j \in \mathbb{N}$. With a single cluster, this can be expressed as:

$$1 = \sum_{j=0}^{\infty} x_j, \tag{10}$$

$$0 = -\lambda x_0 + \mu x_1 - \gamma x_0 \sum_{j=2}^{\infty} x_j, \tag{11}$$

$$\begin{aligned} 0 = &-(\lambda + \mu + \gamma x_0)x_j + \lambda x_{j-1} + \mu x_{j+1} \\ &+ \gamma x_0(x_{2j-1} + 2x_{2j} + x_{2j+1}), \end{aligned} \tag{12}$$

where Equation (12) holds for all $j \geq 1$.

The variation of the number of jobs in steady state is $\sum_{j=0}^{\infty} j\dot{x_j}(t) = 0$. By a direct computation, this leads to $\lambda - \mu \sum_{j=1}^{\infty} x_j = 0$ and using (10), $x_0 = 1 - \lambda/\mu$. Moreover, using Equation (11), $x_1 = \lambda(1-\lambda)\frac{\gamma+\mu}{(1-\lambda)\gamma+\mu^2}$.

Therefore solving the whole system of equations is the same as solving a linear system with free variables $(x_j)_{j\geq 1}$. This system can be rewritten as a matrix Equation $MX = Y$ where M is of the form:

$$M = \begin{bmatrix} m_{11} & m_{12} & \dots & \\ \lambda & m_{22} & m_{23} & \dots \\ 0 & \lambda & m_{33} & \dots \\ 0 & \ddots & \ddots & \ddots \end{bmatrix}, \tag{13}$$

i.e., is triangular plus one line of λ under its diagonal. Let X^0 be the vector defined by $x_1 = 1$ and $x_j = \frac{1}{\mu}\sum_{i=1}^{j-1} m_{ji}x_i$. All the solutions of $M^T X = 0$ can be written αX^0 for some $\alpha \in \mathbb{R}$. Therefore the dimension of the kernel of the matrix M is 1. This shows that there is a unique solution of the system of Equations (10-12).

So far, however, there is no guarantee that this fixed point is non-negative. To prove this, we designed an iteration algorithm over non-negative sequences that converges to the fixed point.

Algorithm 1 Steady-state computation

```
Require: $\lambda, \mu, \gamma$.
  $x_0 \leftarrow 1 - \lambda/\mu$
  $x_1 \leftarrow \lambda(1-\lambda)\frac{\gamma+\mu}{(1-\lambda)\gamma+\mu^2}$
  $\forall j \geq 2 : x_j \leftarrow 0$.
  repeat
    $\forall j \geq 2$
    $x_j \leftarrow \frac{1}{\lambda+\mu+\gamma x_0}\left(\lambda x_{j-1}+\mu x_{j+1}+\gamma x_0(x_{2j-1}+2x_{2j}+x_{2j+1})\right)$
```

Proposition 8. *The successive sequences $(x_j^t)_{j\in\mathbb{N}}$ computed in Algorithm 1 satisfy the following:*

(i) They converge to a sequence $(x_j^\infty)_{j\in\mathbb{N}}$.

(ii) There exists j^* such that x_j^∞ is increasing with j up to $x_{j^*}^\infty$ and is decreasing after.

(iii) $\forall \epsilon > 0$, $\lim_{i\to\infty} x_j^\infty/(\alpha+\epsilon)^j = \lim_{j\to\infty}(\alpha-\epsilon)^j/x_j^\infty = 0$,
where $\alpha = (\lambda+\mu+\gamma x_0 - \sqrt{(\lambda+\mu+\gamma x_0)^2 - 4\mu\lambda})/(2\mu) < 1$.

(iv) x_j^∞ is the only solution of (10)- (12).

This implies that x_j^∞ decreases with an exponential rate: $x_j \approx_{j\to\infty} c\alpha^j$.

Proof. (sketch)
(i) Let $(x_j^t)_{j\in\mathbb{N}}$ be defined by $x_0^t = 1-\lambda$, $\forall t \in \mathbb{N}$, $x_j^0 = 0$ and $x_j^{t+1} = \frac{1}{\lambda+\mu+\gamma x_0}\left(\lambda x_{j-1}^t + \mu x_{j+1}^t + \gamma x_0^t(x_{2j-1}^t + 2x_{2j}^t + x_{2j+1}^t)\right)$.

By induction on t, one can show that x_j^t is positive and increasing in t for all j. Moreover, at each t, there is only a finite number of x_j^t that are non 0 (the first t) and the quantities $y^t \overset{\text{def}}{=} \sum_{j\geq 0} x_j^t$ and $z^t \overset{\text{def}}{=} \sum_{j\geq 1} jx_j^t$ are well defined and finite. The recurrence equation leads to:

$$z^{t+1}(\lambda+\mu+\gamma x_0) = (1-y^t)(\mu-\lambda) + (\lambda+\mu+\gamma x_0)\, z^t \tag{14}$$

Since x^t is increasing in t, z^t is also increasing in t and $1 - z^t \geq 0$. This shows that $x_j^t \leq 1$. Since x_j^t is increasing in t, it converges to some x_j^∞ that satisfies (11)-(12).

(ii) Let j^* be the minimal j such that $x_{j^*}^\infty \geq x_{j^*+1}^\infty$ (it exists since $\sum_{i=0}^\infty x_j \leq 1$ and $\lim_{j\to\infty} x_j = 0$). We define a sequence y_j^t by $y_j^0 = x_j$ for $j \leq j^*$ and $y_j^0 = 0$ for $j > j^*$ and y_j^{t+1} is updated as in Algorithm 1 for $j > j^*$. By a direct induction, one can show that y_j^t is increasing in t and is less than x_j and decreasing for $j \geq j^*$. Therefore it converges to x_j^∞ and x_j^∞ satisfies *(i)*.

(iii) $\mu X^2 - (\lambda+\mu+\gamma x_0)X + \lambda = 0$ has two solutions α and $\bar{\alpha}$ with $0 < \alpha < 1 < \bar{\alpha}$. Moreover, it is positive on $(\alpha; 1)$. This shows that for $\delta \in (\alpha; 1)$ and $j \geq j_\delta$ (j_δ big enough),

$$\delta^i(\lambda+\mu+\gamma x_0) \leq \lambda\delta^{i-1} + \mu\delta^{i+1} + \gamma x_0\left(\delta^{2i-1} + 2\delta^{2i} + \delta^{2i+1}\right).$$

Let us define y_j^t by $y_j^t = x_j$ for $j \leq j_\delta$ and $y_j^0 = x_j\delta^{j-j_\delta}$ for $j > j_\delta$ and y_j^{t+1} is updated as in Algorithm 1 for $j > j_\delta$. y_j^t is decreasing and converges to x_j^∞, showing that $\lim x_j^\infty/(\delta+\epsilon)^j = 0$ for all $\delta > \alpha$ and $\epsilon > 0$. The other limit $\lim(\delta-\epsilon)^j/x_j = 0$ can be proved similarly.

(iv) since $\lim_{j\to\infty} x_j^\infty/(\alpha+\epsilon)^j = 0$, $\sum_{j=1}^\infty x_j^\infty < \infty$. This shows that when t goes to infinity, both parts of Eq. (14) go to a finite limit. This shows that $\sum_{j=0}^\infty x_j^\infty = \lim_t y^t = 1$. □

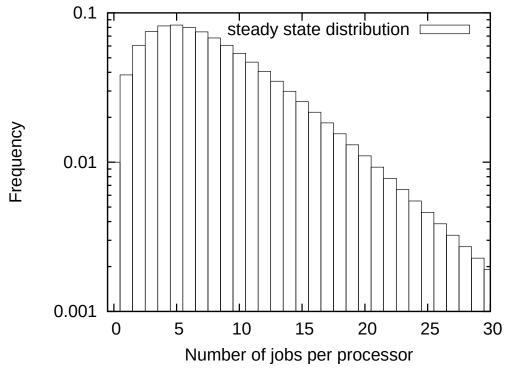


Figure 2: General shape of the steady state distribution. Here for $\lambda = .99$, $\mu = 1$ and $\gamma = 3$. The x-axis is the number of jobs and the y-axis shows the fraction of processors with j jobs (in log-scale).

The shape of the fixed point computed by Algorithm 1 is displayed in Figure 2. *A posteriori*, this can be seen as a justification to consider the ODEs without boundaries since the solution is concentrated almost entirely on small values of j. For all the numerical simulations, we used $K = 500$ and the values of x_{cK} have always been less than 10^{-10}.

It is beyond the scope of this paper to show that this unique fixed point is also a unique attractor of the ODEs (although this is what our numerical experiments suggest). Therefore, our only assertion is that, whenever the steady state distribution of the finite stochastic system converges to a single point, it should converge to this fixed point.

4.2 Gap between mean field and steady-state

In this section, we study two extreme values of γ for which we are able to compute exactly the steady state distribution and compare these results with the mean field approximation.

When $\gamma = 0$, no stealing ever occurs and the N processors behave like independent M/M/1/K queues. The steady state distribution Π_0 has a product form. In steady state, the probability of having j jobs in one M/M/1/K queue is $\pi_j = (\frac{\lambda}{\mu})^j \pi_0$ and satisfies the Equations (10-12). The steady state of the whole cluster is made of N independent variables picked according to this distribution. The law of large number shows that Π_0 converges to x almost surely and the central limit theorem shows that the speed of convergence is in $O(1/\sqrt{N})$. Also, the marginal of the distribution of the steady state for one processor is the mean field steady state.

When $\gamma = \infty$, the system can also be replaced by a simpler one. In that case, there is either no idle processor or no processor with 2 or more jobs since in that case an idle processor would instantly steal the second job. Thus the state of the system can be modeled by the total number of jobs in the system that we call $j^N(t)$. Moreover, $j^N(t)$ behaves like one $M/M/N/KN$ queue (*i.e.*, a queue with N independent processors with arrival rate $N\lambda$ and service rate μ for each processor). The probability that $j^N(t)$ is j is:

$$\begin{aligned}\Pi_\infty(j) &= \Pi_\infty(0)\tfrac{N^j\lambda^j}{j!\mu^j} \quad (j < N),\\ \Pi_\infty(j) &= \Pi_\infty(0)\tfrac{N^j\lambda^j}{N!N^{j-N}\mu^j} \quad (KN \geq j \geq N),\end{aligned}$$

where $\Pi_\infty(0)$ is a normalization constant.

As N grows, it can be shown that $P(j^N(t) \geq N) = o(\beta^N)$ and $\Pi_\infty(0) = \exp(-N\lambda) + o(\beta^n)$ with $(\beta = \lambda \exp(1-\lambda) \in (0;1))$. Let us compute the characteristic function $\Phi()$ of the steady state distribution of $(j^N(t) - N\lambda)/\sqrt{N}$:

$$\Phi(\xi) = \sum_{j=0}^{\infty} \Pi_\infty(j) \exp(i\xi \frac{j - N\lambda}{\sqrt{N}}) = \exp(-\lambda\xi^2 + O(\frac{1}{N})).$$

Therefore $\sqrt{N}(j^N(t)/N - \lambda/\mu)$ converges to a Gaussian law. In that case, the gap between the steady state of the system of finite N and the mean field is of order $1/\sqrt{N}$.

For both extremal cases $\gamma = 0$ and $\gamma = \infty$, the gap between the mean field approximation and the real steady state is of order $O(1/\sqrt{N})$. We conjecture that this should also be the case for all γ in between.

4.3 Average sojourn time

The first set of experiments given in Figure 3 measure the effect of the cost of stealing, $1/\gamma$, on the average sojourn time of the jobs (the average time spent by a job in the system).

Let $S_\lambda(\gamma)$ be the average sojourn time of a job in the limit system (in steady-state). By Little's formula, the number of jobs $L_\lambda(\gamma)$ verifies $L_\lambda(\gamma) = \lambda S_\lambda(\gamma)$, therefore it suffices to compute the average number of jobs in steady state. As mentioned in the previous part, the analytical computation of the steady-state is impossible to do when N is large. This shows that we cannot compute analytically the whole curve $S_\lambda(\gamma)$, nevertheless we can compute two interesting points for $\gamma = 0$ and $\gamma = \infty$.

When $\gamma = 0$, the system is just a system of N independent $M/M/1/K$ queues and the average sojourn time is a classical result of queuing theory (*e.g.*, [3]). When K is large, this is approximately

$$S_\lambda(0) = \frac{1}{\mu - \lambda}. \tag{15}$$

The second quantity that we can compute is the limit of $S_\lambda(\gamma)$ when γ goes to infinity. In that case the steady state is composed of processors with either 0 or 1 job. A new job entering in the system either arrives directly on an empty processor or arrives in an occupied processor and is immediately stolen by an empty processor (there are empty processors with probability one), and the average sojourn time is

$$S_\lambda(\infty) = \frac{1}{\mu}, \tag{16}$$

and the average number of jobs in the system is $L_\lambda(\infty) = \frac{\lambda}{\mu}$.

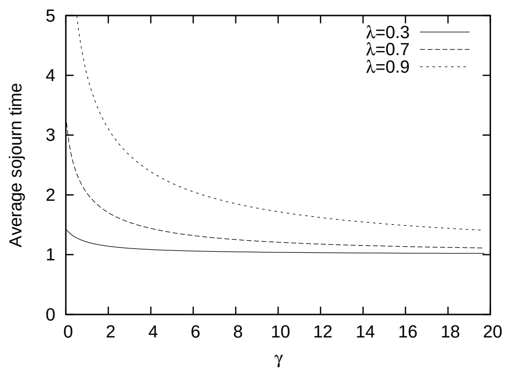


Figure 3: Average sojourn time as a function of the rate of stealing γ for various values of λ (.3, .7 and .9).

Figure 3 displays the average sojourn time $S_\lambda(\gamma)$ as a function of γ for various values of λ. As expected, the average sojourn time is decreasing from $1/(1-\lambda)$ to $1/\mu = 1$. In particular, we can see that when γ is small, the average number of jobs in the system decreases drastically.

The gap between $S_\lambda(0)$ and $S_\lambda(\infty)$ is $\lambda/(1-\lambda)$. Therefore the gain obtained by work stealing is more important when the system is more congested (*i.e.*, λ is close to 1) as we can see in Figure 3. To provide a better estimate of the gain from using work stealing, Figure 4 shows the difference between the response time with a finite γ and $\gamma = \infty$ $S_\lambda(\gamma) - 1$ divided by the maximum gain $S_\lambda(0) - S_\lambda(\infty)$. One can observe that when the rate of stealing is of the same order of the service rate $\gamma = 1$ (resp. $\gamma = 4$ or $\gamma = 8$), the gain is 50% (resp. 80% or 90%) of what one could gain with a work stealing at no cost. This makes work stealing very efficient in real life systems since the time to steal a job is typically small compared to the service time of a job.

4.4 Average number of steals

We want to study the average number of steals that a job undergoes, as a function of γ. Similar to the previous case, let $S_\lambda(\gamma)$ be the average number of steals per job. Again, it is impossible to solve the problem analytically but we can

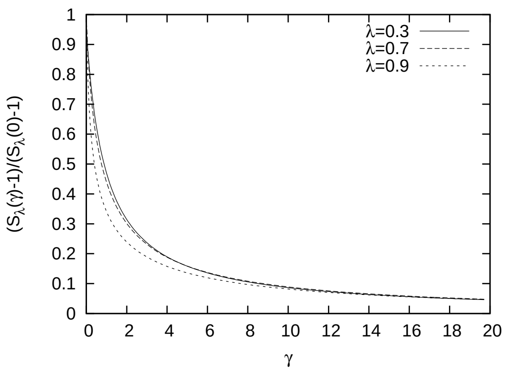


Figure 4: Gain of work stealing as a function of γ.

compute the two extremal values for $\gamma = 0$ and $\gamma = \infty$. When $\gamma = 0$, no job is stolen and $S_\lambda(0) = 0$. When $\gamma = \infty$, all jobs arriving in the processors with 1 job are stolen while the jobs entering in the processors with 0 processors are not stolen. Thus $S_\lambda(\infty) = \lambda/\mu$.

For $0 < \gamma < \infty$, we can compute $S_\lambda(\gamma)$ numerically. The rate of steals from processors with j jobs is $\gamma x_0 x_j$ (where x_j corresponds to the number of processors with j jobs in steady state); each steal corresponding to $\lfloor j/2 \rfloor x_j$ jobs stolen. On average there are $\gamma x_0 \sum_{j=2}^{\infty} \lfloor j/2 \rfloor x_j$ jobs stolen per unit of time. Since the rate of arrivals is λ, we have

$$S_\lambda(\gamma) = \frac{\gamma}{\lambda} x_0 \sum_{j=2}^{\infty} \lfloor \frac{j}{2} \rfloor x_j.$$

Figure 5 shows that the number of steals increases when the cost of stealing decreases and converges to λ, as expected.

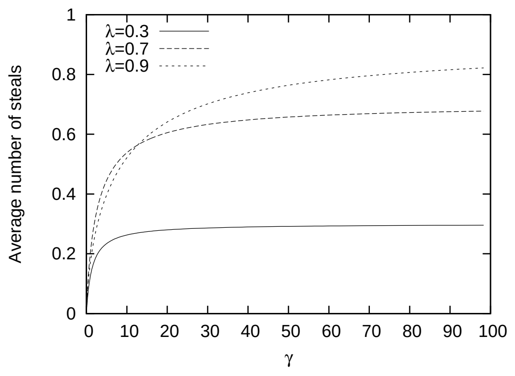


Figure 5: Average number of successful steal per job $S_\lambda(\gamma)$ viewed as a function of γ for different values of λ (.3, .7 and .9).

4.5 Distribution of the number of steals and sojourn time

In many practical cases, the average response time is not a good measure of performance and it is more important to improve the probability of being under a certain threshold. Using our fast simulation algorithm introduced in Section 3.3, we are able to sample the distribution of the number of steals and of the sojourn time in the steady state. Our simulations show that work stealing is indeed efficient at reducing the probability of having a large sojourn time.

To compute the distribution of sojourn times, we have to specify the order in which the jobs are served as well as which jobs are stolen when there is a steal. We consider that the jobs are served in the FCFS order (First Come First Served). When there is a steal, the stealing processor steals the oldest jobs (except for the one that is being served) and the order of the jobs in the processor is preserved *i.e.*, if the jobs in the first processor are $\{1, 2 \dots j\}$ then after the steal the remaining jobs in the victim processor will be $\{1, 2 + \lfloor j/2 \rfloor \dots j\}$ and the jobs in the stealing processor will be $\{2 \dots \lfloor j/2 \rfloor + 1\}$.

Let us now consider a job arriving in the system. Let size_of_queue and place_in_queue be respectively the size of the queue and the place in the queue of this job, and let us consider the next event. If this event is an arrival (respectively a departure), then size_of_queue is increased by 1 (resp. both size_of_queue and place_in_queue are decreased by 1). If the event is a steal, then if place_in_queue $\in [2 \dots \lfloor$size_of_queue/2$\rfloor]$ then the variable size_of_queue becomes $\lfloor$size_of_queue/2$\rfloor$ and place_in_queue decreases by 1. Otherwise, size_of_queue becomes $\lceil$size_of_queue/2$\rceil$ and the variable place_in_queue is decreased by $\lfloor$size_of_queue/2$\rfloor$. The three events occur respectively with rates $\lambda, \mu, \gamma x_0$.

Using these considerations, the sojourn time of a job entering the system as well as the number of steals can be simulated by Algorithm 2.

Algorithm 2 Sojourn time simulation

```
Require: $\lambda, \mu, \gamma$
  Pick size_of_queue according to the steady state distribution.
  $x_0 \leftarrow 1 - \lambda/\mu$.
  size_of_queue $\leftarrow$ size_of_queue + 1
  place_in_queue $\leftarrow$ size_of_queue.
  soj_time $\leftarrow 0$.
  while place_in_queue $> 0$ do
    soj_time $\leftarrow$ soj_time + Exp$(\lambda + \mu + \gamma x_0)$
    Pick an event $e \in \{$arrival, departure, steal$\}$ with probabilities proportional to $\{\lambda, \mu, \gamma x_0\}$ respectively.
    Modify place_in_queue and size_of_queue according to the event $e$.
  end while
  Return soj_time.
```

We ran several simulations for various values of λ and γ. The percentage of jobs that undergo two steals or more is displayed in Figure 6 as a function of γ. Notice that in all cases, the distribution of the number of steals is mostly concentrated in 0 or 1 (less than 2% of the jobs are subjected to more than two steals). Moreover, the shape of the curve is the same for all λ: it starts from 0 when $\gamma = 0$; quickly reaches its maximum and then decreases slowly as γ goes to infinity. The main consequence of this is that there are few useless steals (if a job is stolen twice, then the first steal was actually useless).

Another interesting measure is the sojourn time distribution. When $\gamma = 0$ (*i.e.*, in the M/M/1/K case), the sojourn time distribution has an exponential distribution of parameter $\mu(1 - \lambda/\mu)$ when K goes to infinity. When γ goes to infinity, the sojourn time distribution follows an exponential law of parameter μ. Using our fast simulation algorithm, we can also sample the distribution of the sojourn time.

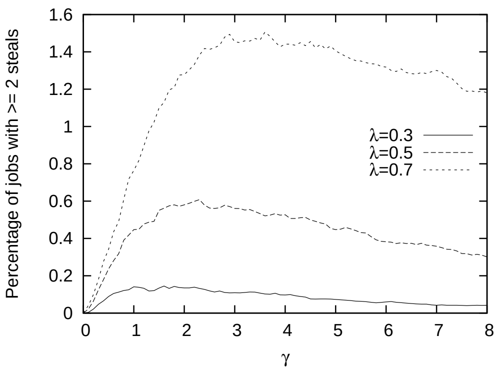


Figure 6: Fraction of jobs that are stolen twice or more as a function of γ.

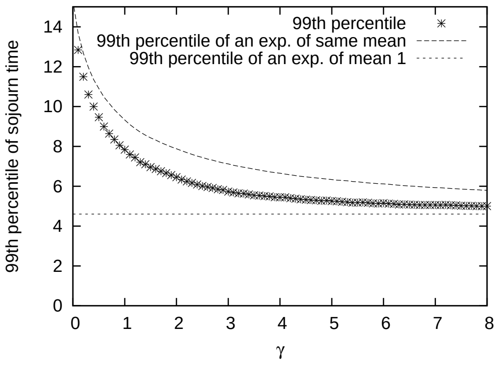


Figure 7: 99 percentiles of the distribution of sojourn time and of an exponential variable of the same mean as functions of γ, for $\lambda = .7$.

The sojourn time of a job in the system is denoted $T_\lambda(\gamma)$. When $\gamma = 0$ or $\gamma = \infty$, the distribution of $T_\lambda(\gamma)$ is an exponential distribution. Therefore for these values of γ, the 99th percentile is $\log(100)S_\lambda(\gamma)$. Figure 7 reports the empirical 99th percentile of the distribution of $T_\lambda(\gamma)$ as well as the 99th percentile of an exponential variable of the same mean $\log(100)S_\lambda(\gamma)$. For intermediate values of γ, the percentile is strictly less than for exponential distributions.

4.6 Fraction of stolen jobs

In all of our analysis, we choose to consider that if there were $j \geq 2$ jobs in a queue, the processor that is stealing would steal $\lfloor j/2 \rfloor$ jobs. This is the most natural strategy in the sense that it optimizes the balance between the processors. Works such as [14] study the case where every steal concerns only one job. This has a negative impact on performance, as shown in the following experiments. In this section, we further show the advantage of stealing half of the jobs instead of stealing a smaller fraction of the work.

When an empty processor steals from a processor with $j \geq 2$ jobs, then it steals $\max(1, \lfloor \delta j \rfloor)$, *i.e.*, a fraction $0 \leq \delta < 1$ of the total work or a single job if this fraction is less than one. The case $\delta = 0$ corresponds exactly to the steal of one job while the condition $\delta < 1$ insures that we always leave at least one job in the processor (since $j - \lfloor \delta j \rfloor) \geq 1$).

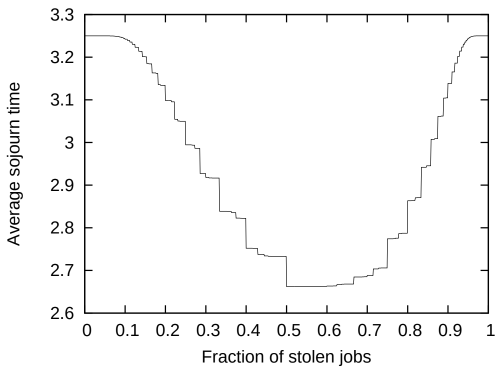


Figure 8: Average sojourn time as a function of the fraction of jobs stolen at each time for $\lambda = .9$ and $\gamma = 3$.

Algorithm 1 can be modified to compute the steady-state in that case. Figure 8 shows the average sojourn time as a function of δ for $\lambda = .9$, $\mu = 1$ and $\gamma = 3$.

Figure 8 highlights two interesting properties. The first one is that the optimal fraction to steal is one half and the difference in speed is approximately 25%. The second interesting property is that this curve is not symmetric in 1/2 and it is not continuous in δ. Indeed, in such systems the processors generally have a small number of jobs and the function $\lfloor \delta j \rfloor$ is not continuous in j. In particular, the figure shows discontinuities at p/q for small values of q $(1/2, 1/3, 2/3 \dots)$.

4.7 Batch arrivals

In this section, we consider that jobs arrive in batches of b jobs according to a Poisson point process of rate λ/b (the rate is divided by b to keep an arrival rate λ). We will see that work stealing is a very efficient way to diminish the effect of these batch arrivals.

In a system with no load balancing mechanism, when the size b of the batches increases, it can be shown that the average sojourn time grows linearly in b [15].

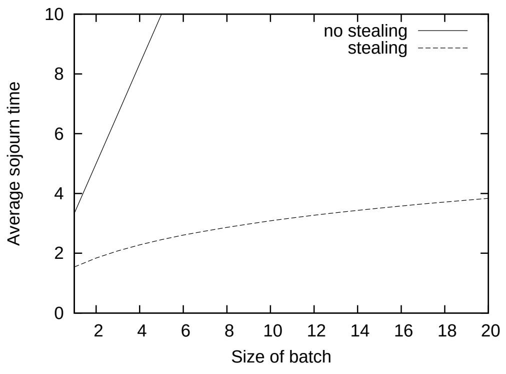


Figure 9: Average sojourn time as a function of the batch size for $\lambda = .7$. The higher curve represents a system without work stealing while the bottom one shows the results for $\gamma = 3$.

Algorithm 1 can be easily modified to take into account the batch arrivals. Using this, we can compute the average sojourn time for various values of the size of batches (from

1 to 30) for $\lambda = .7$ and $\gamma = 3$. In Figure 9 we compare the average sojourn time in the system with work stealing to the system without work stealing. Once again, work stealing has a tremendous impact on the performance. In fact when computing the average sojourn time for large values of b, it seems to grow as the logarithm of b. This is in agreement with known results on the performance of work stealing in transient cases (*i.e.*, over a finite number of tasks) [19].

5. HETEROGENEOUS CLUSTERS

As mentioned in the introduction, we are interested in evaluating the performance of work stealing in a system made of several clusters. Each cluster is made of homogeneous processors with the same processing rate. Also the time to steal between two clusters depends on their "distance" and is much larger than the time to steal within one cluster. The main problem addressed here is to come up with a stealing strategy (namely, values for the stealing probabilities p_{ij}) that minimizes the response times for jobs.

If the system is heterogeneous, the natural work stealing algorithm which is to steal uniformly at random among all the resources may suffer from communication cost since it is much faster to steal inside a cluster. Several modifications of this algorithm have been proposed to take into account the geometry of the system. Many of the proposed algorithms rely on a master-worker paradigm: some resources, called masters, are dedicated to balance the load between clusters while the others try to balance the work insider a cluster. Other strategies are based on tuning the preferences between internal and external steals. All these strategies are experimentally compared in [21, 17] and the latter proved more efficient. In our framework, this corresponds to tuning the probabilities $p_{cc'}$ for choosing a victim.

In the following, we focus on the *average sojourn time* as the performance indicator to compare different strategies. We used a numerical algorithm to compute the fixed point of the system of ODEs (1-9). The time to generate one curve is less than ten minutes, allowing us to explore many scenarios.

5.1 Two homogeneous clusters

A first case of interest is the case with two homogeneous clusters: $\mu_0 = \mu_1$, $\lambda_0 = \lambda_1$ and $\gamma_{00} = \gamma_{11}$. Each cluster can be viewed as a network of closely interconnected processors. The two clusters are connected by a slow network. Generally, communication between two hierarchies (intra-processor, intra-cluster, inter-cluster) is about 10 to 100 times slower (as in grid 5000 [1]). For our simulations, $\gamma_{ij}{=}\gamma_{ii}/10$ for $i{\neq}j$.

Let p_{ij} be the probability for a processor in cluster i to choose to steal from cluster j. As the system is symmetric, we choose $p_{00} = p_{11}$ and $p_{01} = p_{10} = 1 - p_{00}$. We want to study the effect of this probability p_{00} on the performance.

The loads of the two clusters are the same and communications are much slower if the two processors are in two different clusters. Therefore the optimal p_{00} is 1: it is always better to steal inside one's own cluster. Figure 10 displays the average sojourn time of a job entering the system as a function of p_{00}, called the probability of *self-stealing.* The parameters of the system displayed in the figure are $\lambda = .7$, $\mu = 1$ and $\gamma_{ii} = 10, \gamma_{ij} = 3$ and the clusters have the same size; the results are very similar for other values.

This figure exhibits the two main features of such systems. First, as expected in this case, it is much more efficient to pick $p_{00} = 1$ rather than a uniform stealing probability,

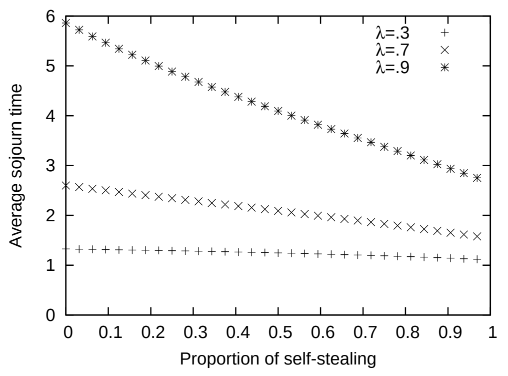


Figure 10: Average sojourn time in a system with two homogeneous clusters as a function of the probability for a processor to steal inside its cluster.

$p_{00} = 1/2$. Moreover, the dependence of the sojourn time on p_{00} is almost linear, showing that this probability has a real effect. Computing the same curve for other values of λ, shows that when the load is low (for example $\lambda = .3$), then the curve is slightly concave while when the load is very high ($\lambda > .9$), the curve is slightly convex but in all cases the dependence is almost linear.

5.2 Two heterogeneous clusters

A direct extension of the previous model is to consider the case where the two clusters are heterogeneous. We set $\lambda_0 < \lambda_1$, $\mu_0 = \mu_1$ and $\gamma_{ij} = \gamma_{ii}/10$. As the load of cluster 1 is higher and since it is faster to steal inside the cluster, we consider that the processors of cluster 1 only steal inside their cluster: $p_{11}{=}1$. We want to study the effect of the probability for the processors in cluster 0 to steal from cluster 1.

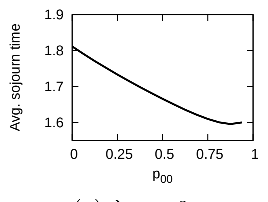
Figure 11 displays the average sojourn time as a function of p_{00} for different values of λ_1. In all cases, the load of cluster 0 is low: $\lambda_0 = .5$ and the load of the other cluster varies. In all cases, we can see that there exists an optimal p_{00} that is neither 0 nor 1 that minimizes the average sojourn time. In the top left figure, cluster 1 is just slightly more loaded than the cluster 0 ($\lambda_1 = .8$) and in that case the optimal p_{00} is close to .85. As the load of the cluster 1 grows, the optimal probability gets closer to 0.

This shows that the optimal probability strongly depends on the load of the different clusters which is an unknown variable in many cases. However, we also see that the average sojourn time does not vary that much around the optimal p_{00} which shows that a rough estimation of the load is enough to make a good choice for p_{00}.

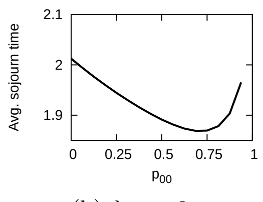
5.3 Hierarchical work stealing: master worker paradigm

Many work stealing algorithms rely on a master/worker paradigm. Here we show that this approach is indeed valid, but tuning its parameters is not easy.

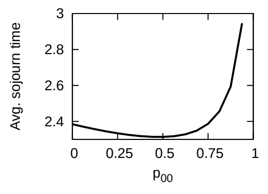
We consider a network of homogeneous clusters, with a rate of steal 10 times greater inside a cluster than between two clusters. In each cluster, we set a fraction f_0 of the resources to be "masters" while the rest of the resources are "workers". We consider that the masters receive all the work which means that there is an arrival rate of λ/f_0 for every master (so that the total arrival rate of the system remains



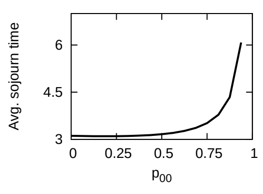
(a) $\lambda_1 = .8$



(b) $\lambda_1 = .9$



(c) $\lambda_1 = 1$



(d) $\lambda_1 = 1.1$

Figure 11: Average sojourn time as a function of p_{00} for the two heterogeneous model. The first cluster is lightly loaded ($\lambda_0 = .5$). The load of the second cluster is λ_1 (varying from $.8$ to 1.1).

constant equal to λ). A master will only steal work from other masters while a worker can steal both from a master and from other workers.

5.3.1 Master-worker in one cluster

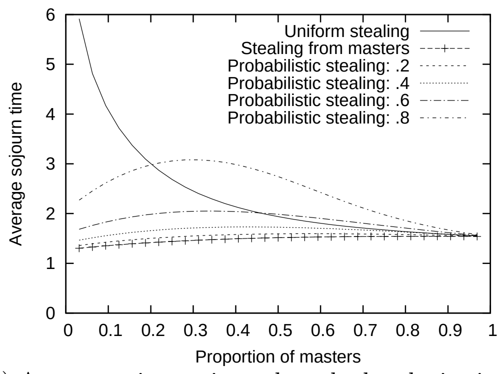
Let us first consider a network composed of only one cluster and let us study the effect of the fraction of masters on the performance of the system. We compare three different strategies: *probabilistic stealing* with various parameters, *uniform stealing* and *steal from masters.* This situation can be described in our model by setting the number of clusters to 2, cluster 0 representing the masters and cluster 1 representing the workers. We set $\gamma_{ij} = \gamma_{ii}$, $\lambda_0 = \lambda/f_0$, $\lambda_1 = 0$ and $p_{00} = 1$. For the probabilistic stealing strategy, we study different probabilities for a worker to steal a master $p_{10} = .2, .4, .6, .8$. The *uniform stealing* (resp. *steal only from masters*) corresponds to $p_{10} = x_0$ (resp. $p_{10} = 1$). We also compare the case where the jobs arrive one by one and the case of batch arrivals with a batch size of 20.

The average sojourn time for all strategies as a function of the fraction of masters is shown in Figure 12. As expected, in all cases, the uniform stealing is the worst strategy when the number of masters is low and when the proportion of masters grows, things get better. When the proportion of masters is close to 1, all strategies coincide. When the batches are of size 1, the optimal strategy is always to steal from the masters. When the batches are of size 20, the optimal strategy is to steal about 50% from the masters (more precisely, the optimal is about 60% for low proportions of masters ($< .3$) and 40% above).

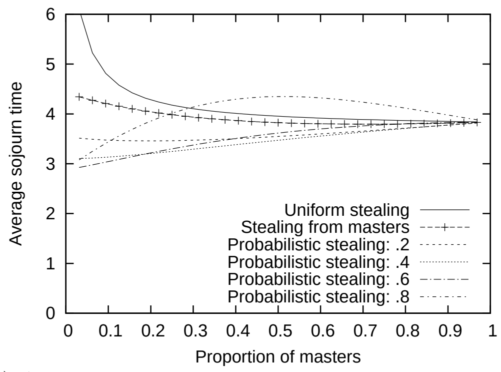
The most interesting property shown in this figure is that in all cases, having all the arrivals concentrated on a few resources improves the performance of the system if we tune the probability of stealing correctly.

5.3.2 Master-worker in two clusters

The behavior observed in the two cluster model is similar to the behavior with a single cluster. We consider a network of two clusters with the same proportion f_0 of masters in each cluster. All the arrivals are concentrated on the masters. The masters are only stealing from other masters and we



(a) Average sojourn time when the batch size is 1.



(b) Average sojourn time when the batch size is 20.

Figure 12: Comparison of the average sojourn time in the Master-Worker setting with one cluster.

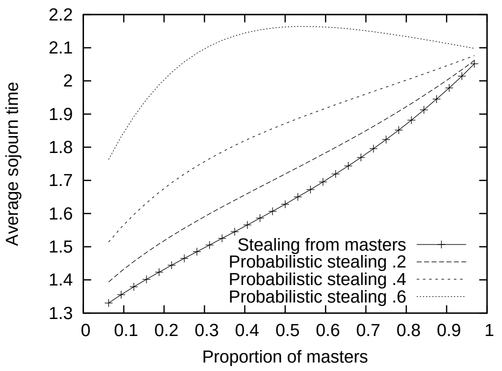
study different strategies of stealing for the workers. As in the one cluster case, the uniform policy is not efficient. Therefore, we only focus on the policies *stealing from master* and *probabilistic stealing* with a probability of .2, .4 and .6.

The results are shown in Figure 13, comparing a case with arrivals by batches of size 20 and a case without batch arrivals. Once again we see that in the case where the batches are of size 1, the most efficient strategy is always to steal from the masters while if the size of the batches is larger (20 in this example), the most efficient strategy is to steal from workers with a non zero probability. Moreover in both cases, a good choice of the probabilities makes the performance of the system better when the proportion of masters is low.

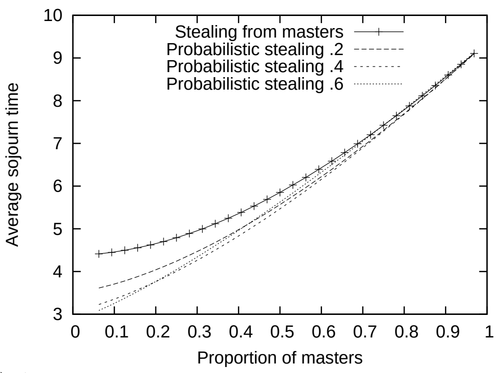
When the proportion of masters is low, the good performance can be explained in part by the fact that we neglect the congestion that might occur when a single resource has to deal with too arrivals or steal requests. In practical applications, one would have to take this into account.

6. CONCLUSION AND FUTURE WORK

In this paper we presented a mean field approximation of the work stealing algorithm on a large number of processors and show convergence for finite time as well as for steady state. This allows one to run a rather exhaustive evaluation for several performance measures such as average response times, average number of steals per job. The distribution of response times can also be sampled using fast simulation, providing more meaningful performance indexes (variance,



(a) Average sojourn time when the batch size is 1.



(b) Average sojourn time when the batch size is 20.

Figure 13: Average sojourn time in the Master-Worker setting with two clusters for $\lambda = .7$.

percentiles, tail behavior). This also allows one to evaluate work stealing under several scenarios: all processors are in one homogeneous cluster, or partitioned into several clusters where the time to steal from one cluster to another depends on the distance between the clusters. The scenario where work is allocated to a few master processors and stolen by workers is also evaluated and numerical evidences show that it performs really well with appropriate stealing parameters.

We are currently considering two extensions of this work. The first one is to model the case with a finite number of jobs using a mean approach, and then tune the parameters to minimize the total completion time. The second one is to conduct a thorough comparison with push and pull policies currently used in computational grids.

Acknowledgement We are grateful to Pr. Martin Arlitt whose insightful comments substantially improved this paper.

7. REFERENCES

[1] Grid 5000, a large scale nation wide infrastructure for grid research. https://www.grid5000.fr/.

[2] J. Anselmi and B. Gaujal. Performance evaluation of a work stealing algorithm for streaming applications. In *13th Int. Conf. On Principles Of DIstributed Systems (OPODIS)*, Nîmes, 2009.

[3] F. Baccelli and P. Bremaud. *Elements of queueing theory.* Springer-Verlag, 2003.

[4] M. Benaim. Recursive algorithms, urn processes and chaining number of chain recurrent sets. *Ergodic Theory and Dynamical Systems*, 18(01):53–87, 1998.

[5] M. Bender and M. Rabib. Online scheduling of parallel programs on heterogeneous systems with applications to cilk. *Theory of Computing Systems*, 35:289–304, 2002. Special issue on SPA00.

[6] P. Berenbrink, T. Friedetzky, and L. Goldberg. The natural work-stealing algorithm is stable. *SIAM J. Comput.*, 32(5):1260–1279, 2003.

[7] J. Bernard, J-L. Roch, and D. Traore. Processor-oblivious parallel stream computations. In *16th Euromicro Int. Conf. on Parallel, Distributed and network-based Processing*, Toulouse, 2008.

[8] S. Ethier and T. Kurtz. *Markov processes: Characterization and convergence.* Wiley, 1986.

[9] M. Frigo, C. Leiserson, and K. Randall. The implementation of the cilk-5 multithreaded language. In *ACM SIGPLAN'98 Conf. on Programming Language Design and Implementation*, volume 33, pages 212–223, Montreal, 1998.

[10] T. Gautier, X. Besseron, and L. Pigeon. Kaapi: A thread scheduling runtime system for data flow computations on cluster of multi- processors. In *Proc. of the 2007 int. workshop on Parallel symbolic computation, PASCO'07*, pages 15–23, N.Y., 2007.

[11] O. Kallenberg. *Foundations of modern probability.* Springer Verlag, 2002.

[12] A. Kukanov and M. Voss. The foundations for scalable multi-core software in intel threading building blocks. *Intel Technology Journal*, 11(4):309–322, 2007.

[13] T. Kurtz. *Approximation of population processes.* Society for Industrial Mathematics, 1981.

[14] M. Mitzenmacher. Analyses of load stealing models based in differential equations. In *10th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 212–221, 1998.

[15] A. Müller and D. Stoyan. *Comparison methods for stochastic models and risks.* Series in Probability and Statistics. Wiley, 2002.

[16] M. Norman. A central limit theorem for Markov processes that move by small steps. *The Annals of Probability*, 2(6):1065–1074, 1974.

[17] J.-N. Quintin and F. Wagner. Hierarchical work stealing. Technical Report 7077, INRIA, 2009.

[18] M. Squillante and R. Nelson. Analysis of task migration in shared-memory multiprocessor scheduling. *SIGMETRICS Perf. Eval. Rev.*, 19(1):143–155, 1991.

[19] M. Tchiboukdjian, D. Trystram, J.-L. Roch, and J. Bernard. List Scheduling: The Price of Distribution. Research Report 7208, INRIA, 2010.

[20] H. Tembine, J.-Y. Le Boudec, R. El-Azouzi, and E. Altman. Mean Field Asymptotic of Markov Decision Evolutionary Games and Teams. *GameNets*, 2009.

[21] R. Van Nieuwpoort, T. Kielmann, and H. Bal. Efficient load balancing for wide-area divide-and-conquer applications. *ACM SIGPLAN*, 36(7):34–43, 2001.

[22] R. van Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H. Bal. Satin: Simple and efficient java-based grid programming. In *AGridM Workshop on Adaptive Grid Middleware*, New Orleans, 2003.